Evolutionary Algorithms-assisted Construction of Cryptographic Boolean Functions

Claude Carlet
University of Bergen
Bergen, Norway
claude.carlet@gmail.com

Domagoj Jakobovic
University of Zagreb
Zagreb, Croatia
domagoj.jakobovic@fer.hr

Stjepan Picek
Delft University of Technology
Delft, The Netherlands
s.picek@tudelft.nl

ABSTRACT

In the last few decades, evolutionary algorithms were successfully applied numerous times for creating Boolean functions with good cryptographic properties. Still, the applicability of such approaches was always limited as the cryptographic community knows how to construct suitable Boolean functions with deterministic algebraic constructions. Thus, evolutionary results so far helped to increase the confidence that evolutionary techniques have a role in cryptography, but at the same time, the results themselves were seldom used.

This paper considers a novel problem using evolutionary algorithms to improve Boolean functions obtained through algebraic constructions. To this end, we consider a recent generalization of Hidden Weight Boolean Function construction, and we show that evolutionary algorithms can significantly improve the cryptographic properties of the functions. Our results show that the genetic algorithm performs by far the best of all the considered algorithms and improves the nonlinearity property in all Boolean function sizes. As there are no known algebraic techniques to reach the same goal, we consider this application a step forward in accepting evolutionary algorithms as a powerful tool in the cryptography domain.

CCS CONCEPTS

• **Security and privacy** → *Mathematical foundations of cryptography*; • **Mathematics of computing** → **Evolutionary algorithms**; • **Computing methodologies** → **Genetic algorithms**; **Genetic programming**;

KEYWORDS

Boolean function, Cryptography, Secondary Construction, Hidden Weight Boolean Function

ACM Reference Format:
Claude Carlet, Domagoj Jakobovic, and Stjepan Picek. 2021. Evolutionary Algorithms-assisted Construction of Cryptographic Boolean Functions. In *2021 Genetic and Evolutionary Computation Conference (GECCO '21), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3449639.3459362

GECCO '21, July 10–14, 2021, Lille, France

ACM ISBN 978-1-4503-8350-9/21/07…$15.00
https://doi.org/10.1145/3449639.3459362

1 INTRODUCTION

Evolutionary computation (EC) represents an interesting option (often as a last-resort option) for many difficult problems. Common examples include scheduling [11], cancer detection [2], and communications networks [15]. One more domain where evolutionary computation proved to be useful is the security domain. There, we can find a plethora of diverse applications, like in fuzzing [31], side-channel attacks [33], and evolution of cryptographic primitives like Boolean functions [26] and pseudorandom number generators [29]. Interestingly, it seems that the evolution of Boolean functions took most of the evolutionary computation community interest. There are several intuitive reasons for this: 1) Boolean functions are easy to encode and evolve, 2) there are multiple interesting properties, which gives numerous interesting scenarios, and 3) evolutionary computation can reach top performance for evolving Boolean function, where this performance rivals the results of the algebraic construction. Unfortunately, this also means that evolutionary computation is commonly unable to give better results than algebraic constructions (with rare exceptions like [20]), which makes this line of research interesting but not applicable in practice. Differing from this, we propose a novel application of evolutionary computation for the evolution of Boolean functions. More precisely, we call our approach EC-assisted construction of Boolean functions, where we use evolutionary computation to improve the results obtained through algebraic constructions.

Improving the results of algebraic constructions is a relevant problem as we can obtain significantly better results. This becomes especially important in cases where algebraic constructions do not provide sufficiently good results. Improving algebraic construction results is also a difficult problem as commonly, we do not know any deterministic technique to improve the results. While exhaustive search could be considered a viable option to evaluate whether improvements are possible, exhaustive search is not practically possible. Indeed, in cryptography, one commonly works with large Boolean functions with huge search space (in general, for a Boolean function of n inputs, there are 2^{2^n} possible Boolean functions).

There are no results considering evolutionary computation to help construct Boolean functions with good cryptographic properties to the best of our knowledge. Several works use evolutionary computation to produce Boolean functions to be used in algebraic constructions or to evolve algebraic constructions [22, 27]. The closest approach we found would probably be using evolutionary computation to evolve addition chains that are then used in public-key cryptography [21].

In this work, we consider a recent proposal of a generalized Hidden Weight Boolean Function (HWBF) construction [7], based on a general construction of so-called parameterized Boolean functions [5]. This construction is efficient to implement and results in good cryptographic properties. While the properties are good, they are far that can be obtained with other constructions like Carlet-Feng [8], which are, on the other hand, very computationally complex and are then not really usable in practical stream ciphers, since these need to be lighter and faster than block ciphers and the Boolean function used as filter being the only nonlinear part of the cipher, the complexity, and speed of the whole cipher lies precisely in the Boolean function. The generalized HWBF construction can be (easily) improved by switching bits in the function's truth table. Unfortunately, many possible positions can be swapped, and no mathematical results determine how many bits to flip (beyond the fact that if we want to improve the nonlinearity by an additive factor δ, we know that we need to change at least δ bits) or on what positions. Since random search does not give good results and exhaustive search is computationally infeasible for practical sizes, we must look for different approaches. This paper proposes an evolutionary approach where we experiment with several evolutionary algorithms and Boolean function sizes. Our results show that genetic algorithms work the best and provide significantly better nonlinearity than the Hidden Weight Boolean Function (or its generalization). We consider this to be the first work in the EC-assisted construction of Boolean functions in cryptography.

2 TECHNICAL BACKGROUND

2.1 Notation

Let n be a positive integer, i.e., $n \in \mathbb{N}^+$. The set of all n-tuples of elements in the field $\mathbb{F}_2$ is denoted as $\mathbb{F}_2^n$ where $\mathbb{F}_2$ is the Galois field with two elements. We denote the inner product of two vectors a and b by $a \cdot b$; it equals $a \cdot b = \bigoplus_{i=0}^{n-1} a_i b_i$, with "$\oplus$" being the addition modulo two (bitwise XOR). The support ($supp$) of a Boolean function f is the set containing the non-zero positions in the truth table representation, i.e., $supp(f) = \{x : f(x) = 1\}$. The Hamming weight $w_H(f)$ of a Boolean function f equals the size of its support. For $u = (u_1, \ldots, u_n), v = (v_1, \ldots, v_n) \in \mathbb{F}_2$, we define the partial order on $\mathbb{F}_2^n$ as $u \preceq v$ if and only if $u_i \leq v_i, \forall i$.

2.2 Boolean Functions

An $(n, 1)$-function is any mapping f from $\mathbb{F}_2^n$ to $\mathbb{F}_2$ and such a function is called a Boolean function. A Boolean function f on $\mathbb{F}_2^n$ can be uniquely represented by a truth table, which is a vector $(f(0), \ldots, f(1))$ that contains the function values of f with inputs ordered lexicographically, i.e., $a \leq b$. The Walsh-Hadamard transform W_f is a unique representation of a Boolean function that measures the correlation between $f(x)$ and the linear functions $a \cdot x$ [6]:

$$W_f(a) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) \oplus a \cdot x}. \tag{1}$$

A Boolean function f is *balanced* if it takes the value 1 exactly the same number 2^{n-1} of times as the value 0 when the input ranges over $\mathbb{F}_2^n$. If the function is imbalanced, it is not suitable for usage in cryptography as one can attack its biased output.

The minimum Hamming distance between a Boolean function f and all affine functions (in the same number of variables as f) is called the nonlinearity of f. The nonlinearity Nl_f of a Boolean function f can be expressed in terms of the Walsh-Hadamard coefficients as [6]:

$$Nl_f = 2^{n-1} - \frac{1}{2} \max_{a \in \mathbb{F}_2^n} |W_f(a)|. \tag{2}$$

The nonlinearity of a Boolean function with n inputs is bounded above as follows

$$Nl_f \leq 2^{n-1} - 2^{\frac{n}{2}-1}. \tag{3}$$

This bound is usually called the Covering Radius Bound (this inequality is an equality for so-called bent functions, which exist for n even only). When n is odd, the bound given in Eq. (3) cannot be tight. Then, the maximal nonlinearity lies between $2^{n-1} - 2^{\frac{n-1}{2}}$ and $2^{n-1} - 2^{\frac{n}{2}-1}$.

In our evolutionary assisted construction, we will concentrate on only those two properties: balancedness and nonlinearity. Still, we are interested in an additional cryptographic property called algebraic immunity (AI) [10]. Since this property is computationally expensive (e.g., one evaluation for $n = 16$ lasts several hours, we do not include it in the evolution process. We conducted a posteriori tests of algebraic immunity and found that the results were good (i.e., on the HWBF construction level). Finally, we note that in cryptography (and more precisely, in the design of stream ciphers), Boolean functions' minimum size (i.e. number of variables, that is, number of input bits) with practical importance is 13 inputs [6]. For additional information about Boolean functions and their cryptology applications, we refer interested readers to [6].

2.3 Hidden Weight Boolean Functions

The Hidden Weight (Weighted) Boolean Function (HWBF) is a Boolean function in n variables defined as follows [4]:

$$f(x) = \begin{cases} 0 & \text{if } x = 0 \\ x_{w_H(x)} & \text{otherwise.} \end{cases}$$

The advantages of HWBF are that it has good algebraic immunity (not optimal, but at least $\lfloor n/3 \rfloor + 1$), it is balanced, and its output is considerably faster to compute than those of the other currently known functions having good algebraic immunity [32]. Unfortunately, HWBF has poor nonlinearity, which makes it not practical to use in cryptography. More precisely, the nonlinearity parameter equals

$$Nl_f = 2^{n-1} - 2\binom{n-2}{\lceil \frac{n-2}{2} \rceil}. \tag{4}$$

Note that the nonlinearity given in Eq. (4) is more or less the worst nonlinearity of all known functions with optimal algebraic immunity.

3 RELATED WORK

As already discussed, evolutionary computation is commonly used to create Boolean functions with good cryptographic properties. Common examples of such works are Millan et al., where the authors apply genetic algorithms to evolve Boolean functions with high nonlinearity [17]. Millan et al. used GA, hill climbing, and a resetting step to evolve Boolean functions with high nonlinearity [18]. In their work, they considered Boolean functions up to

12 inputs. Dawson et al. investigated two-stage optimization to generate Boolean functions [9]. More precisely, they used simulated annealing and hill-climbing with a cost function motivated by the Parseval theorem to find functions with high nonlinearity and low autocorrelation. Kavut and Melek developed improved cost functions for a search that combines simulated annealing and hill climbing [13]. With that approach, the authors were able to find some functions of eight and nine inputs that have a combination of nonlinearity and autocorrelation values previously not obtained. Millan et al. proposed a new adaptive strategy for the local search algorithm for the generation of Boolean functions with high nonlinearity [19]. Hernan et al. used a multi-objective random bit climber to search for balanced Boolean functions of size up to eight inputs with high nonlinearity. [1]. Picek et al. experimented with genetic algorithms and genetic programming to find Boolean functions that fulfill several cryptographic constraints [25]. Mariot and Leporati used Particle Swarm Optimization to find Boolean functions with good trade-offs of cryptographic properties for dimensions up to 12 inputs [16]. Picek et al. conducted a detailed analysis of the efficiency of several evolutionary algorithms and fitness functions for Boolean functions with eight inputs [26]. Picek et al. used immunological algorithms to evolve highly nonlinear Boolean functions with up to 16 inputs [28].

Next, we discuss several works that used evolutionary algorithms in different combinations with algebraic constructions. Picek et al. considered an interesting approach of evolving Boolean functions that is somewhat orthogonal to ours. More precisely, the authors evolved Boolean functions in a certain number of inputs to use them to construct larger Boolean functions algebraically [27]. Picek and Jakobovic used genetic programming to evolve secondary algebraic constructions of bent (maximally nonlinear) Boolean functions [22]. Picek and Jakobovic also investigated how evolutionary algorithms can evolve algebraic constructions of S-boxes (which are vectorial Boolean functions) [23]. For a more detailed overview of evolutionary computation applications for Boolean functions in cryptography, we refer interested readers to [24].

4 PROBLEM DEFINITION

Recently, C. Carlet proposed a generalization of the Hidden Weight Boolean Function that allows a construction of n-variable balanced functions f from $(n-1)$-variable Boolean functions g satisfying some rather light condition, see Eq. (6) below [7]. The function is defined as:

$$f_{F_g}(x) = (x_{w_H(x)+1}+1)(g(x^{'})+1)+x_{w_H(x)}(x)g(x^{''}), x \in \mathbb{F}_2^n. \quad (5)$$

Here, $x^{'}$ is the vector obtained from x by erasing its coordinate of index $w_H(x)+1$ and $x^{''}$ is the vector obtained from x by erasing its coordinate function of index $w_h(x)$.

Function g can be any function fulfilling the property:

$$\forall u \in \mathbb{F}_2^{n-2};\ g(u^{(0)} \leq g(u^{(1)}), \quad (6)$$

where we denote by $u^{(j)}$ the vector obtained from u by inserting a coordinate of value j at position $w_H(u)+1$ (and shifting on the right by one position all the coordinates whose indices were at least $w_H(u)+1$ before the insertion). Many functions are fulfilling this property where obvious examples are monotone Boolean functions.

n	$HWBF$	$GHWBF_3$	Exhaustive	Random
6	20	22	24	24
7	44	46	52	52
8	88	94	104	104
9	186	194	–	216
10	372	394	–	426
11	772	802	–	862
12	1544	1628	–	1710
13	3172	3284	–	3440
14	6344	6668	–	6732
15	12952	13372	–	13674
16	25904	27158	–	27026
17	52666	54250	–	54984
18	105332	110194	–	108320

Table 1: The best-obtained nonlinearity results for the HWBF anf GWHBF functions. The last two columns show nonlinearity of $GHWBF_3$ after applying random search and exhaustive search to improve the nonlinearity. The results include the first five monomials.

A Boolean function is monotone if whenever $u \leq v$, then $f(u) \leq f(v)$.

This generalized HWBF construction (we denote it as GHWBF) allows keeping HWBF quality of being fast to compute if function g is fast enough to compute and having good algebraic immunity while improving its nonlinearity. Note that the construction results in a function f in n variables, and to build it, we use a function g in $n-1$ variables. If g is a constant function 1, then GHWBF becomes HWBF.

As already stated, there are multiple choices for the function g. The current results indicate that the best nonlinearity is obtained when g is a monomial function of the form $\prod_{i=0}^{n-1} x_i$. What is more, the best results are reached when the degree of the monomial equals 3 (i.e., has the form $x_i x_j x_z$). In our experiments, we consider only the monomial functions of degree 3, and we denote the resulting function f_{F_g} as $GHWBF_3$.

Still, this results in a large number of possible monomials one should potentially investigate. The best nonlinearity results obtained through GHWBF are given in Table 1. We also denote the values one reaches with the HWBF function and the results obtained after improving nonlinearity through random search or exhaustive search (for small n) as discussed in the next paragraphs.

Clearly, $GHWBF_3$ improves over the nonlinearity of $HWBF$, but the values are still far from the upper bound as given in Eq. (3). Naturally, the problem arises from the choice of the function g as it is easy to notice that some other functions g that fulfill Eq. (6) reach much lower nonlinearity. Thus, a natural question is whether we can modify the function g to still fulfill Eq. (6) at the expense of, e.g., not being a monotone function anymore. Indeed, C. Carlet discussed that visiting all u and choosing $g(u^{(0)})$ *and* $g(u^{(1)})$ such that $g(u^{(0)}) \leq g(u^{(1)})$, is computationally intractable, but also may result in some contradictions because there are more than one possible choice of such u, given some z. Taking z nonzero and different from the all-1 vector, and writing $z = (z_1, \ldots, z_{w_H(z)}, z_{w_H(z)+1}, \ldots, z_{n-1})$:

(1) if $z_{w_H(z)} = 1$ and $z_{w_H(z)+1} = 0$, then z can be obtained as $u^{(0)}$ for $u = (z_1, \ldots, z_{w_H(z)} - 1, z_{w_H(z)+1}, \ldots, z_{n-1})$ (which has weight $w_H(z) - 1$ and the insertion is correctly made at position $w_H(u) + 1 = w_H(z)$) and as $u^{(1)}$ for $u = (z_1, \ldots, z_{w_H(z)}, z_{w_H(z)+2}, \ldots, z_{n-1})$ (which has weight $w_H(z)$ and the insertion is correctly made at position $w_H(u) + 1 = w_H(z) + 1$). One then needs to make choices on the values taken by g that can avoid contradictions.
(2) if $z_{w_H(z)} = 0$ and $z_{w_H(z)+1} = 1$ then z will never be obtained as $u^{(0)}$ or $u^{(1)}$. Consequently, $g(z)$ can be arbitrary.
(3) if $z_{w_H(z)} = z_{w_H(z)+1}$ then z will be obtained only once as $u^{(0)}$ or $u^{(1)}$ and there is no risk a contradiction.

The most straightforward approach to further improve the nonlinearity results would be to take $GHWBF_3$ and modify them so that they are no more monotone but still satisfy Eq. (6). To do so, we would change the values of g taken at the inputs such that $z_{w_H(z)} = 0$ and $z_{w_H(z)+1} = 1$. Unfortunately, this does not tell us how many such values to change or on what positions exactly. Running the experiments for function f with a small dimension n shows that the function g (which is in $n - 1$ variables) can be changed in $\frac{1}{4}$ of the positions in its truth table. This results in computationally intractable computations for any function size that has practical importance. For instance, if we consider function f in $n = 13$ inputs, this means that the function g has 2^{12} positions in the truth table. This again means that we can change 2^{10} specific values in the truth table. While it is easy to calculate their positions, we cannot know the exact combination of changes and positions to improve the nonlinearity. What is more, it is clear that the exhaustive search (denoted as *Exhaustive* in Table 1) can work for values $n \leq 8$, as already $n = 8$ gives computational complexity of 2^{32} combinations (considering one monomial only). Note that random search (denoted as *Random*) can be used, but it does not give competitive results already for $n = 9$. Interestingly, we can observe that even random search reached higher nonlinearity values than obtained from $HWBF$ and $GHWBF_3$.

5 EXPERIMENTAL SETUP

5.1 Encodings and Algorithms

We consider solutions to be masks of values denoting whether we flip a bit on a certain position or not. As such, we consider two intuitive solution encodings: bitstring and tree encoding. In the bitstring encoding, a bit value equal to 1 means that a bit on that position will be flipped, and a bit value equal to 0 means no change. The length of the encoding is of the same size as the length of the maximal number of positions to swap (2^{n-3}, where n is the size of the Boolean function f). Note that we internally map each of the individual's bit values to the corresponding position that can be swapped in the truth table of function g. We assume lexicographical ordering, where the first bit in the individual solution represents the first bit in g allowed to be swapped, etc.

The second encoding we use is tree encoding. More precisely, we evolve a Boolean function in the form of a syntactic tree by using genetic programming. The truth table of the evolved function is then applied the same way bitstring encoding is used to denote bitflips in function g. Genetic programming (GP) [14] works on the population of computable expressions, where the most common form is symbolic expressions corresponding to parse trees. The building elements in a tree-based GP are functions (inner nodes) and terminals (leaves, problem variables). The terminal set consists of $n - 3$ input Boolean variables, denoted $\{v_0, \ldots, v_{n-3}\}$. The function set (i.e., the set of inner nodes of a tree) should consist of appropriate Boolean operators that allow the definition of any function with n inputs. The function set used in all the experiments consists of Boolean functions OR, XOR, AND (taking two arguments), NOT (one argument), and IF (it takes three arguments and returns the second argument if the first one evaluates to 'true' and the third one otherwise). The application of this particular function set is based on our previous experience in applying GP to the Boolean domain.

Genetic Algorithm (GA). For GA [12], we use a 3-tournament selection, which eliminates the worst individual among three randomly selected ones. After the elimination, a new individual is produced using the crossover operator applied on the remaining two. The new individual immediately undergoes mutation subject to a defined individual mutation rate. The crossover operators are one-point and uniform crossover, performed uniformly at random for each new offspring. The mutation operator is selected uniformly at random between a simple mutation, where a single bit is inverted, and a mixed mutation that randomly shuffles the bits in a randomly selected substring. The individual mutation probability is used to select whether an individual would be mutated or not, and the mutation operator is executed only once on a given individual.

Evolution Strategy (ES). We use $(\mu + \lambda)$-ES [3], where in each generation, parents compete with offspring, and from their joint set, μ test individuals are kept. The offspring is generated using the same mutation operators as used in the GA.

Genetic Programming (GP). In our experiments, GP uses the same steady-state tournament selection algorithm as the GA. The variation operators are simple tree crossover, uniform crossover, size fair, one-point, and context preserving crossover (selected at random), and subtree mutation [30].

5.2 Parameter Tuning

For all algorithms, we conduct a short tuning phase; more precisely, for GA, we explore the population size (200 and 500). For GP, we explore the maximum tree depth (3, 5, and 7) and the population size (200, 500). Following the tuning, for GA and GP we select the population size of 500 and the mutation rate of 0.3, and the GP has the maximum tree depth equal to 5. All experiments are repeated 50 times, and initial populations are created uniformly at random. For all algorithms, as the termination criterion, we use 1 000 000 evaluations or stagnation in 25 000 evaluations.

5.3 Fitness Functions

The first fitness function uses the nonlinearity value with the goal of maximizing it:

$$fitness_1 = Nl_f. \tag{7}$$

In the second fitness function, we consider the Walsh-Hadamard spectrum's values to have more granularity and provide gradient information. Recall that the nonlinearity depends on the highest values in the Walsh-Hadamard spectrum. Thus, observing the spectrum and minimizing the *number* of the highest values could also lower the final nonlinearity. Note that due to the Parseval theorem, the sum of all Walsh-Hadamard values is fixed and equals 2^{2n}, which means lowering some of the spectrum's values increases some other values. Our second fitness function looks at the nonlinearity value but also the whole Walsh-Hadamard spectrum, and we aim to maximize it:

$$fitness_2 = Nl_f + \frac{2^n - count}{2^n}, \tag{8}$$

where *count* denotes the number of times that the Walsh-Hadamard spectrum's largest value is encountered (since the Walsh-Hadamard values can be less than zero, we take the absolute value). As already stated, the maximal value determines nonlinearity and having that value as low as possible will increase the nonlinearity. Note that we normalize the second part of the fitness function in the $[0, 1]$ range so that the actual nonlinearity is still the primary criterion.

6 RESULTS

In this section, we present the experimental results from our investigation. We consider only the first five monomials that reach the nonlinearity equal to that from $GHWBF_3$. Our results indicate very similar behavior (the maximal nonlinearity values after running the improvement experiments) for those monomials, so we believe that adding more monomials to the experiments would not bring significantly different results.

Table 2 gives the best-obtained results for the fitness function 1 and all five monomials (if existing) for all algorithms. The highest nonlinearities are given in bold font. Notice that the nonlinearity values across different monomials are rather similar. As the monomial 1 reached the highest value the most times (10 out of 13 sizes), we continue with a detailed analysis for it only.

For monomial 1, in Table 3 we give detailed results for both fitness functions and all four considered algorithms (GA, ES, GP, random search). The results are aggregated over 50 independent runs where we consider the maximal nonlinearity value reached in every run. The best fitness values are given in bold style. Considering fitness 1 (which equals the nonlinearity value) and sizes 6 to 8, all algorithms behave the same, and in every run, the same maximal value is obtained. For larger sizes, GA reaches the highest nonlinearity, with GP being the second best. On average, ES is the third-best algorithm, and as expected, random search performs the worst. Notice how fitness 2 improves the resulting nonlinearity values for all algorithms except random search. While ES and GP benefit from extra information provided by the fitness function already for Boolean functions with ten inputs, GA manages to reach higher nonlinearity for all Boolean function sizes from 14 to 18 inputs.

Figure 1 depicts the distribution of results for Boolean function sizes 13 to 18. The white circle represents the median value, while the thick line inside the shape denotes quartile 1 and quartile 3. Note that fitness function 2 gives slightly higher values than fitness function 1 due to the additional term in the expression. Still, since that term is limited to $[0, 1]$, we can disregard that term's influence in the graph. The visible differences are the result of differing nonlinearity values. Figures 1a and 1b show significantly different behavior for GA and the rest of the algorithms. First, the GA reaches much higher values, and the results are concentrated around only a few fitness values. For both ES and GP, we notice a much wider spread of obtained values, where for fitness function 1, GP works substantially better than ES. Going to the fitness function 2 improves the *maximal* nonlinearity for ES and GP, but we can notice that GP now has more spread out values and lower minimal value. At the same time, the difference between the maximal values for those two algorithms decreases. Larger Boolean function sizes (Figures 1c and 1d) continue with the trend we described but now, the differences are even more pronounced. For GA, we can observe a clear improvement in the fitness values when going from fitness 1 to fitness 2. For fitness 1 and sizes 15 and larger, GP is significantly better than ES and has a larger spread of obtained values. Going to fitness function 2 shows that the differences between the maximal fitness values for GP and ES are reduced and that GP has a larger spread of values. Notice that this decrease in the difference of the maximal values happens as ES improves its performance, while GP does not seem to benefit from the extra information provided by

n	mon. 1	mon. 2	mon. 3	mon. 4	mon. 5
6	$x_1x_3x_4$ **24**	$x_2x_3x_4$ 22	$x_1x_4x_5$ **24**	$x_3x_4x_5$ **24**	– –
7	$x_1x_2x_4$ **52**	$x_1x_3x_4$ **52**	$x_2x_3x_4$ 50	$x_1x_4x_6$ **52**	$x_2x_4x_6$ **52**
8	$x_1x_4x_5$ **104**	$x_2x_4x_5$ **104**	$x_3x_4x_5$ 102	$x_4x_5x_6$ **104**	$x_4x_5x_7$ **104**
9	$x_1x_4x_5$ **216**	$x_2x_4x_5$ **216**	$x_3x_4x_5$ **216**	$x_4x_5x_7$ **216**	$x_4x_5x_8$ **216**
10	$x_1x_5x_6$ **438**	$x_2x_5x_6$ **438**	$x_3x_5x_6$ **438**	$x_4x_5x_6$ 434	$x_5x_6x_8$ **438**
11	$x_1x_5x_6$ **888**	$x_2x_5x_6$ **888**	$x_3x_5x_6$ **888**	$x_4x_5x_6$ 886	$x_5x_6x_8$ **888**
12	$x_1x_6x_7$ **1794**	$x_2x_6x_7$ **1794**	$x_3x_6x_7$ **1794**	$x_4x_6x_7$ **1794**	$x_5x_6x_7$ 1780
13	$x_1x_6x_7$ **3614**	$x_2x_6x_7$ **3614**	$x_3x_6x_7$ 3612	$x_4x_6x_7$ 3612	$x_5x_6x_7$ 3606
14	$x_1x_7x_8$ 7266	$x_2x_7x_8$ 7266	$x_3x_7x_8$ **7270**	$x_4x_7x_8$ 7268	$x_5x_7x_8$ **7270**
15	$x_1x_7x_8$ **14612**	$x_2x_7x_8$ 14610	$x_3x_7x_8$ **14612**	$x_4x_7x_8$ **14612**	$x_5x_7x_8$ 14606
16	$x_1x_8x_9$ 29162	$x_2x_8x_9$ 29166	$x_3x_8x_9$ 29168	$x_4x_8x_9$ 29184	$x_5x_8x_9$ **29196**
17	$x_1x_8x_9$ 58734	$x_2x_8x_9$ **58762**	$x_3x_8x_9$ 58724	$x_4x_8x_9$ 58732	$x_5x_8x_9$ **58762**
18	$x_1x_9x_{10}$ **116840**	$x_2x_9x_{10}$ 116796	$x_3x_9x_{10}$ 116810	$x_4x_9x_{10}$ 116804	$x_5x_9x_{10}$ 116838

Table 2: The best obtained nonlinearities for the first five monomials, GA/ES/GP, and fitness function 1. Monomial 1 reaches the highest nonlinearity the most times. For $n = 6$, there are only four monomials satisfying the conditions for the nonlinearity of $GHWBF_3$ so there are no results to display.

	GA				ES				GP				Random			
n	min	max	average	stdev	min	max	average	stdev	min	max	average	stdev	min	max	average	stdev
								Fitness 1								
6	24	**24**	24.0	0.0	24	**24**	24.0	0.0	24	**24**	24.0	0.0	24	**24**	24.0	0.0
7	52	**52**	52.0	0.0	52	**52**	52.0	0.0	52	**52**	52.0	0.0	52	**52**	52.0	0.0
8	104	**104**	104.0	0.0	102	**104**	103.9	0.5	102	**104**	103.8	0.7	102	**104**	102.4	0.8
9	216	**216**	216.0	0.0	214	**216**	214.4	0.8	212	216	214.0	1.0	210	214	212.0	0.7
10	436	**438**	437.6	0.8	426	430	428.8	1.1	426	432	428.7	1.5	418	424	420.3	1.8
11	886	**888**	887.9	0.4	868	878	875.1	2.1	870	880	875.4	2.8	846	862	850.7	3.3
12	1788	**1794**	1791.2	1.2	1738	1754	1745.6	4.0	1734	1758	1747.3	5.5	1668	1700	1676.7	6.6
13	3608	**3614**	3611.2	1.2	3514	3560	3541.5	9.9	3514	3564	3547.9	10.4	3386	3412	3395.4	6.0
14	7242	**7266**	7256.6	4.7	7020	7102	7061.7	19.3	7026	7106	7073.9	17.0	6672	6722	6688.1	10.4
15	14570	**14612**	14594.8	8.7	14146	14292	14236.8	37.4	14144	14414	14320.1	69.7	13614	13666	13634.3	11.2
16	29066	**29162**	29112.7	22.4	28172	28500	28359.4	86.8	28072	28744	28608.3	116.3	26814	26960	26858.7	27.0
17	58574	**58734**	58681.7	25.9	56844	57340	57178.5	132.5	56684	57878	57429.8	365.8	54790	54922	54856.4	30.6
18	116682	**116840**	116779.2	34.7	113028	113194	113091.5	36.1	113832	115688	115224.0	391.4	108096	108264	108171.4	41.5
								Fitness 2								
6	24.8	**24.8**	24.8	0.0	24.8	**24.8**	24.8	0.0	24.8	**24.8**	24.8	0.0	24.8	**24.8**	24.8	0.0
7	52.9	**52.9**	52.9	0.0	52.9	**52.9**	52.9	0.0	52.9	**52.9**	52.9	0.0	52.9	**52.9**	52.9	0.0
8	105	**105**	105.0	0.0	105	**105**	105.0	0.0	105	**105**	105.0	0.0	103	**105**	103.5	0.9
9	217	**217**	217.0	0.0	217	**217**	217.0	0.0	213	**217**	215.7	1.3	211	215	212.8	0.8
10	437	**439**	438.9	0.4	431	435	433.0	1.4	427	435	430.9	1.8	419	425	422.0	1.4
11	889	**889**	889.0	0.0	877	885	882.4	2.2	869	883	877.8	3.4	847	859	852.2	2.6
12	1791	**1795**	1793.4	1.1	1743	1767	1754.0	5.5	1735	1761	1748.9	6.0	1667	1695	1677.5	5.2
13	3609	**3615**	3613.0	1.3	3529	3571	3557.0	8.1	3509	3575	3549.2	15.1	3385	3413	3397.5	6.6
14	7255	**7275**	7266.7	4.9	7033	7097	7077.1	12.5	6971	7109	7074.8	25.6	6671	6715	6690.4	10.0
15	14611	**14639**	14624.4	7.4	14237	14403	14360.8	27.8	14031	14421	14331.7	62.8	13615	13693	13634.7	13.6
16	29233	**29343**	29309.2	17.0	28543	28681	28600.1	34.9	28067	28723	28576.9	126.0	26809	26951	26858.2	27.5
17	58861	**58935**	58920.6	19.3	57385	57613	57553.1	57.5	56839	58009	57591.8	312.6	54799	54907	54851.2	26.8
18	116909	**117021**	116963.5	45.5	113385	113503	113448.0	36.5	113743	115703	114820.0	660.4	108099	108305	108178.7	46.2

Table 3: Fitness values for the first monomial. All values are rounded to one decimal place. Sizes 8 to 11 differ from the third decimal place, while larger sizes differ from the fifth decimal place only. From the min and max columns of fitness function 2, it is easy to obtain the nonlinearity: 1) for every value that has decimal part, remove the decimal part, and 2) for every value that is odd, subtract 1.

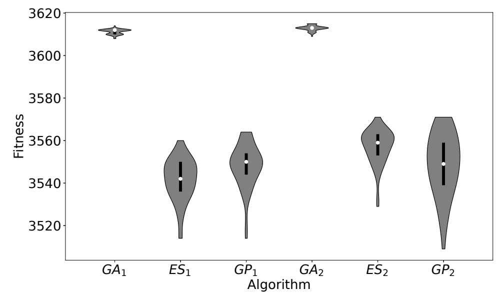
the fitness function 2. We believe this is because for GP, evolving the masks that improve the nonlinearity value is a difficult problem as we do not assume how many inputs the mask needs to change, so GP tries to find a suitable mask while using all terminals.

In Figure 1d, we depict the results for $n = 16$, where we confirm our assumptions for GP: extra information does not benefit the nonlinearity significantly but only changes the frequency of specific nonlinearity values that are reached. ES improves the most considering the change from fitness function 1 to fitness function 2, but those values are still far from GA (where we also see an improvement due to fitness function 2). Finally, in Figures 1e and 1f, we depict the results for sizes 17 and 18, respectively. It is obvious how the differences in the performance between GA and other algorithms continue to increase. For both GA and ES, we observe substantial benefit from using fitness function 2, while for GP, the differences are small (but still in favor of the fitness function 2).

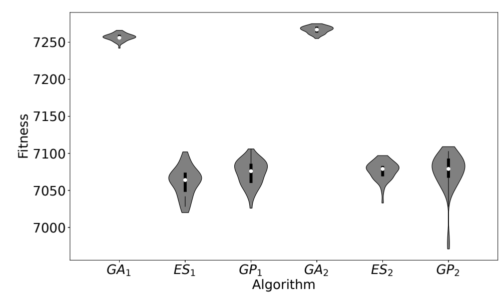
Next, in Figure 2, we depict the convergence results for sizes 13 to 18 and both fitness functions. Here, we confirm our previous observations as we see that fitness function 2 results in higher values, where the most pronounced differences are for ES. On the other hand, GP shows little to no improvement (there are somewhat more pronounced differences for sizes 17 and 18). Interestingly, we see that GA and GP have a rapid increase in the fitness values and smaller improvements after 40% to 50% of evaluations. ES shows a slower increase, but it does not seem to reach stagnation even after 100% of evaluations. ES works with a much smaller number of solutions at a time, and it requires significantly more evaluations to reach the same fitness levels as with other algorithms. Still, we observe that additional information provided in fitness function 2 is highly beneficial and enables ES to reduce the difference from the GP's performance.

Based on the previous observations, we can give several general remarks:

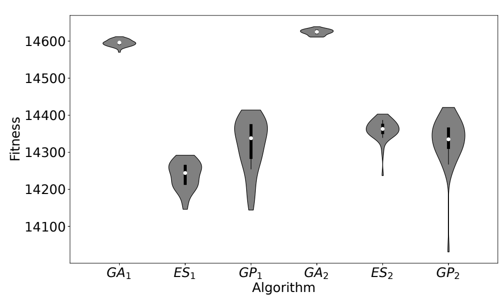
(1) Extra information in the fitness function about the Walsh-Hadamard spectrum improves the performance of EAs.
(2) GA works by far the best, where even fitness function 1 works better than fitness function 2 for other algorithms.
(3) GP works better than ES if considering the first fitness function. These differences diminish with the second fitness function as the performance of ES improves dramatically (the improvements for GP are more modest, and for smaller sizes, fitness function 2 mostly results in a wider spread of obtained values). For sizes 17 and 18, ES does not reach good performance even with the second fitness function.
(4) ES converges slower than other algorithms and could need significantly more evaluations to approach the performance of GP (and especially GA).
(5) GP works surprisingly poorly considering the results from related works on Boolean functions' evolution with good cryptographic properties. We hypothesize this is partly because GP tries to construct a single Boolean function in $n-3$ variables. However, the intermediate function properties



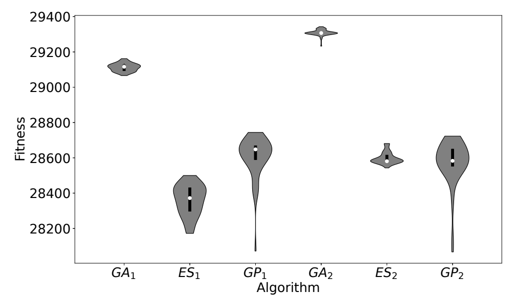
(a) Boolean function with 13 inputs



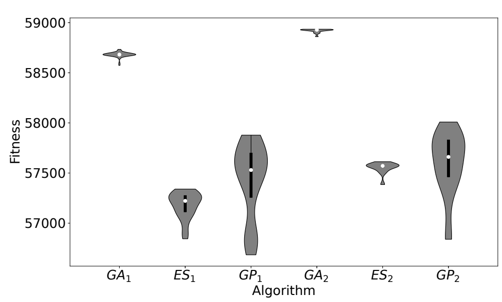
(b) Boolean function with 14 inputs



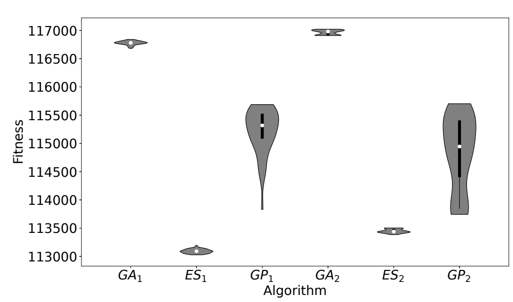
(c) Boolean function with 15 inputs



(d) Boolean function with 16 inputs



(e) Boolean function with 17 inputs



(f) Boolean function with 18 inputs

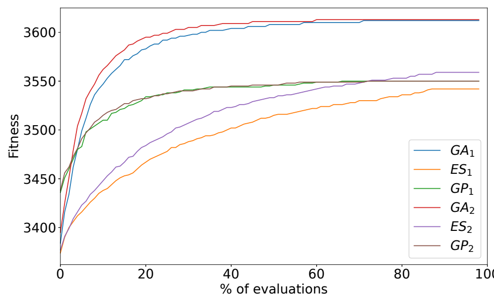
Figure 1: Violin plots for two fitness functions and three evolutionary algorithms. We denote fitness function as a subscript in the algorithm name, e.g., GA_1 represents GA and fitness function 1.

have little relevance since its truth table is only used as a bit flip mask of nonadjacent positions in the truth table of another Boolean function (g) of $n-1$ variables. Any structure of the intermediate function of $n-3$ variables that GP creates (the genotype) is disrupted in the decoding into the resulting function g (the phenotype).

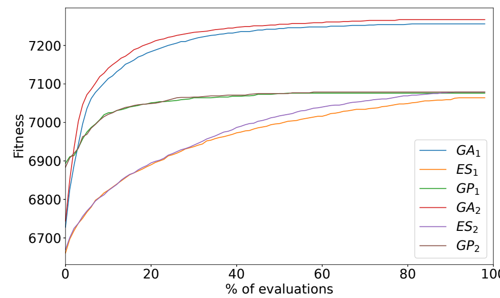
(6) Unfortunately, the results for all algorithms are still rather far from the upper bound for nonlinearity.

7 CONCLUSIONS AND FUTURE WORK

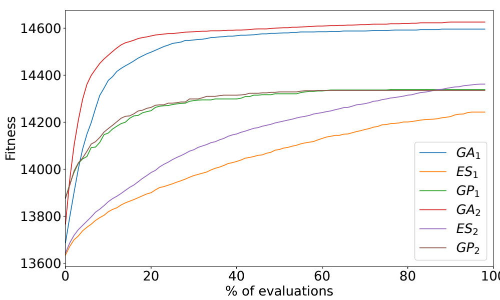
In this paper, we approach the practical problem of improving Boolean functions' nonlinearity values obtained through algebraic constructions. To this end, we develop a novel technique we call evolutionary-assisted construction of Boolean functions, and we test it on the recently proposed generalization of the Hidden Weight Boolean Function construction. Our results indicate that evolutionary algorithms (most notably, GA) significantly improve the nonlinearity values for all investigated Boolean function sizes, where the largest differences can be observed for Boolean functions with more inputs that also have practical relevance. Additionally, we



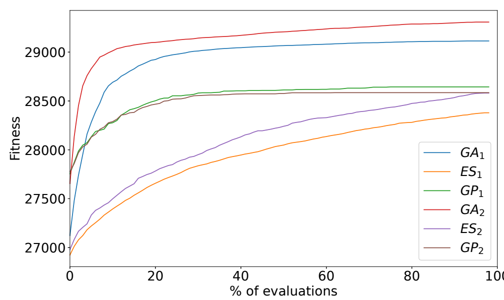
(a) Boolean function with 13 inputs



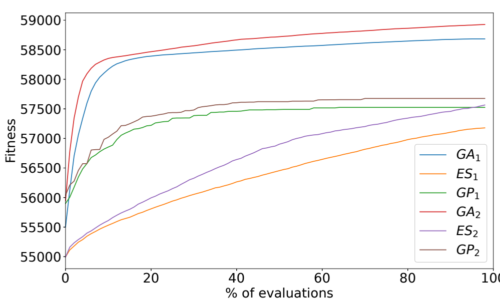
(b) Boolean function with 14 inputs



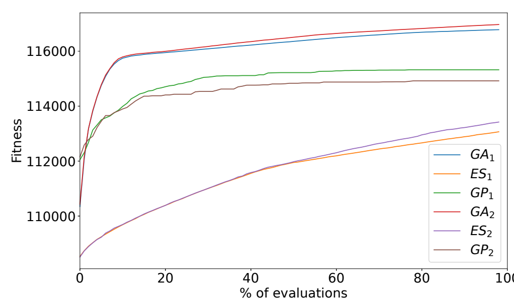
(c) Boolean function with 15 inputs



(d) Boolean function with 16 inputs



(e) Boolean function with 17 inputs



(f) Boolean function with 18 inputs

Figure 2: Convergence graphs for two fitness functions and three evolutionary algorithms

can observe that GA enables larger nonlinearity improvements over $GHWBF_3$, than the increase in $GHWBF_3$ over $HWBF$. We consider our results to be especially important as there are no other known (and practical) techniques to help improve the nonlinearity value of Boolean functions obtained through the generalized HWBF construction.

Our experiments started from construction solutions obtained through the monomial functions of degree 3 as this construction gave the best results among all the tested ones. As EA managed to improve the nonlinearity values obtained with $GHWBF_3$ considerably, it would be interesting to explore other functions g. We leave as open question whether it is possible to improve the nonlinearity value even more if starting with the less fit Boolean functions. Finally, we mentioned that we also require good values of algebraic immunity (AI). Since AI evaluation is expensive, we do not consider it in our fitness function in the hope that the best-obtained results will have good AI. Nevertheless, we could evaluate AI for smaller values of n (e.g., up to 10) and gain insights if the increase in AI is possible.

ACKNOWLEDGMENTS

The research of Claude Carlet is partly supported by the Trond Mohn Foundation.

REFERENCES

[1] Hernan Aguirre, Hiroyuki Okazaki, and Yasushi Fuwa. 2007. An Evolutionary Multiobjective Approach to Design Highly Non-linear Boolean Functions. In *Genetic and Evolutionary Computation Conference (GECCO)*. 749–756.

[2] Qurrat Ul Ain, Harith Al-Sahaf, Bing Xue, and Mengjie Zhang. 2020. A genetic programming approach to feature construction for ensemble learning in skin cancer detection. In *GECCO '20: Genetic and Evolutionary Computation Conference, Cancún Mexico, July 8-12, 2020*, Carlos Artemio Coello Coello (Ed.). ACM, 1186–1194. https://doi.org/10.1145/3377930.3390228

[3] Hans-Georg Beyer and Hans-Paul Schwefel. 2002. Evolution Strategies –A Comprehensive Introduction. *Natural Computing: An International Journal* 1, 1 (May 2002), 3–52. https://doi.org/10.1023/A:1015059928466

[4] R. E. Bryant. 1991. On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. *IEEE Trans. Comput.* 40, 2 (1991), 205–213. https://doi.org/10.1109/12.73590

[5] Claude Carlet. 2020. Parametrizing Boolean functions by vectorial functions and studying related constructions. (December 2020). https://www.math.univ-paris13.fr/~carlet/english.html.

[6] Claude Carlet. 2021. *Boolean Functions for Cryptography and Coding Theory*. Cambridge University Press. https://doi.org/10.1017/9781108606806

[7] Claude Carlet. 2021. A class of Boolean functions generalizing the hidden weight bit function. (March 2021). https://www.math.univ-paris13.fr/~carlet/english.html.

[8] Claude Carlet and Keqin Feng. 2008. An Infinite Class of Balanced Functions with Optimal Algebraic Immunity, Good Immunity to Fast Algebraic Attacks and Good Nonlinearity. In *Advances in Cryptology - ASIACRYPT 2008*, Josef Pieprzyk (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 425–440.

[9] John A Clark and Jeremy L Jacob. 2000. Two-Stage Optimisation in the Design of Boolean Functions. In *Information Security and Privacy*. LNCS, Vol. 1841. Springer, 242–254.

[10] Nicolas T. Courtois and Willi Meier. 2003. Algebraic Attacks on Stream Ciphers with Linear Feedback. In *Advances in Cryptology — EUROCRYPT 2003*, Eli Biham (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 345–359.

[11] Marko Durasevic and Domagoj Jakobovic. 2020. Comparison of schedule generation schemes for designing dispatching rules with genetic programming in the unrelated machines environment. *Appl. Soft Comput.* 96 (2020), 106637. https://doi.org/10.1016/j.asoc.2020.106637

[12] John H. Holland. 1992. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press. https://doi.org/10.7551/mitpress/1090.001.0001

[13] Selçuk Kavut and Melek D. Yücel. 2003. Improved Cost Function in the Design of Boolean Functions Satisfying Multiple Criteria. In *Progress in Cryptology - INDOCRYPT 2003*. LNCS, Vol. 2904. Springer, 121–134.

[14] John R. Koza. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

[15] David Lynch, Takfarinas Saber, Stepán Kucera, Holger Claussen, and Michael O'Neill. 2019. Evolutionary learning of link allocation algorithms for 5G heterogeneous wireless communications networks. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*, Anne Auger and Thomas Stützle (Eds.). ACM, 1258–1265. https://doi.org/10.1145/3321707.3321853

[16] Luca Mariot and Alberto Leporati. 2015. Heuristic Search by Particle Swarm Optimization of Boolean Functions for Cryptographic Applications. In *Genetic and Evolutionary Computation Conference, GECCO, Companion Material Proceedings*. 1425–1426.

[17] William Millan, Andrew Clark, and Ed Dawson. 1997. An Effective Genetic Algorithm for Finding Highly Nonlinear Boolean Functions. In *First Int. Conference on Information and Communication Security (ICICS '97)*. Springer, 149–158.

[18] William Millan, Andrew Clark, and Ed Dawson. 1998. Heuristic Design of Cryptographically Strong Balanced Boolean Functions. In *Advances in Cryptology - EUROCRYPT '98*. 489–499.

[19] William Millan, Joanne Fuller, and Ed Dawson. 2004. New concepts in evolutionary search for Boolean functions in cryptology. *Computational Intelligence* 20, 3 (2004), 463–474.

[20] Stjepan Picek, Claude Carlet, Sylvain Guilley, Julian F. Miller, and Domagoj Jakobovic. 2016. Evolutionary Algorithms for Boolean Functions in Diverse Domains of Cryptography. *Evolutionary Computation* 24, 4 (2016), 667–694. https://doi.org/10.1162/EVCO_a_00190

[21] Stjepan Picek, Carlos A. Coello Coello, Domagoj Jakobovic, and Nele Mentens. 2016. Evolutionary Algorithms for Finding Short Addition Chains: Going the Distance. In *Evolutionary Computation in Combinatorial Optimization*, Francisco Chicano, Bin Hu, and Pablo García-Sánchez (Eds.). Springer International Publishing, Cham, 121–137.

[22] Stjepan Picek and Domagoj Jakobovic. 2016. Evolving Algebraic Constructions for Designing Bent Boolean Functions. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016 (GECCO '16)*. Association for Computing Machinery, New York, NY, USA, 781–788. https://doi.org/10.1145/2908812.2908915

[23] Stjepan Picek and Domagoj Jakobovic. 2019. On the Design of S-Box Constructions with Genetic Programming. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '19)*. Association for Computing Machinery, New York, NY, USA, 395–396. https://doi.org/10.1145/3319619.3322040

[24] Stjepan Picek and Domagoj Jakobovic. 2020. Evolutionary Computation and Machine Learning in Cryptology. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion (GECCO '20)*. Association for Computing Machinery, New York, NY, USA, 1147–1173. https://doi.org/10.1145/3377929.3389886

[25] Stjepan Picek, Domagoj Jakobovic, and Marin Golub. 2013. Evolving Cryptographically Sound Boolean Functions. In *Genetic and Evolutionary Computation Conference (GECCO) (GECCO '13 Companion)*. ACM, 191–192.

[26] Stjepan Picek, Domagoj Jakobovic, Julian F. Miller, Lejla Batina, and Marko Cupic. 2016. Cryptographic Boolean functions: One output, many design criteria. *Appl. Soft Comput.* 40 (2016), 635–653.

[27] Stjepan Picek, Elena Marchiori, Lejla Batina, and Domagoj Jakobovic. 2014. Combining Evolutionary Computation and Algebraic Constructions to Find Cryptography-Relevant Boolean Functions. In *Parallel Problem Solving from Nature – PPSN XIII*, Thomas Bartz-Beielstein, Jürgen Branke, Bogdan Filipič, and Jim Smith (Eds.). Springer International Publishing, Cham, 822–831.

[28] Stjepan Picek, Dominik Sisejkovic, and Domagoj Jakobovic. 2017. Immunological algorithms paradigm for construction of Boolean functions with good cryptographic properties. *Eng. Appl. of AI* 62 (2017), 320–330.

[29] Stjepan Picek, Dominik Sisejkovic, Vladimir Rozic, Bohan Yang, Domagoj Jakobovic, and Nele Mentens. 2016. Evolving Cryptographic Pseudorandom Number Generators. In *Parallel Problem Solving from Nature (PPSN)*. Springer, 613–622.

[30] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. 2008. *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd.

[31] Sanjay Rawat, Vivek Jain, Ashish Kumar, Lucian Cojocar, Cristiano Giuffrida, and Herbert Bos. 2017. VUzzer: Application-aware Evolutionary Fuzzing.. In *NDSS*, Vol. 17. 1–14. https://doi.org/10.14722/ndss.2017.23404

[32] Qichun Wang, Claude Carlet, Pantelimon Stănică, and Chik How Tan. 2014. Cryptographic properties of the hidden weighted bit function. *Discrete Applied Mathematics* 174 (2014), 1 – 10. https://doi.org/10.1016/j.dam.2014.01.010

[33] Zhenbin Zhang, Liji Wu, An Wang, Zhaoli Mu, and Xiangmin Zhang. 2015. A Novel Bit Scalable Leakage Model Based on Genetic Algorithm. *Sec. and Commun. Netw.* 8, 18 (Dec. 2015), 3896–3905. https://doi.org/10.1002/sec.1308